LPI exam 201 prep: System maintenance

Intermediate Level Administration (LPIC-2) topic 211

Skill Level: Intermediate

David Mertz, Ph.D. (mertz@gnosis.cx)
Developer
Gnosis Software

02 Sep 2005

In this tutorial, David Mertz begins preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 201. In this sixth of eight tutorials, you learn basic concepts of system logging, software packaging, and backup strategies.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at junior and intermediate levels. To attain each level of certification, you must pass two LPI exams.

Each exam covers several topics, and each topic has a weight. The weights indicate the relative importance of each topic. Very roughly, expect more questions on the exam for topics with higher weight. The topics and their weights for LPI exam 201 are:

Topic 201
Linux kernel (weight 5).

Topic 202
System startup (weight 5).

Topic 203
Filesystem (weight 10).

Topic 204
Hardware (weight 8).

Topic 209
File and service sharing (weight 8).

Topic 211
System maintenance (weight 4). The focus of this tutorial.

Topic 213
System customization and automation (weight 3).

Topic 214
Troubleshooting (weight 6).

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "System maintenance," the sixth of eight tutorials designed to prepare you for LPI exam 201. In this tutorial, you learn basic concepts of system logging, software packaging, and backup strategies.

The tutorial is organized according to the LPI objectives for this topic, as follows:

2.211.1 System logging (weight 1)
You will be able to configure syslogd to act as a central network log server. This objective also includes configuring syslogd to send log output to a central log server, logging remote connections, and using grep and other text utilities to automate log analysis.

2.211.2 Packaging software (weight 1)
You will be able to build a package. This objective includes building (or rebuilding) both RPM and DEB packaged software.

2.211.3 Backup operations (weight 2)
You will be able to create an offsite backup storage plan.

This tutorial, like the corresponding LPI exam, is a grab-bag of several topics that do not cleanly fall into other categories. System logging and analyzing log files are important tasks for a system administrator to be familiar with. Likewise, a maintained system should carry out a sensible backup strategy using standard Linux tools.

Not every system administrator will need to create custom software packages, but administrators of multiple (similar) installations will need to install site- or company-specific software packages as part of their duties. This tutorial looks at the

Debian and RPM package formats, and touches on basic "tarballs."

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Section 2. System logging

About logging

Many processes and servers under a Linux system append changing status information to "log files." These log files often live in the /var/log/ directory and often begin with a time stamp indicating when the described event occurred. But for better or worse, there is no real consistency in the precise format of log files. The one feature you can pretty much count on is that Linux log files are plain ASCII files, and contain one "event" per line of the file. Often (but not always) log files contain a (relatively) consistent set of space- or tab-delimited data fields.

Some processes, especially Internet services, handle log file writes within their own process. At heart, writing to a log file is just an append to an open file handle. But many programs (especially daemons and `cron`'d jobs) use the standard syslog API to let the syslogd or klogd daemons handle the specific logging process.

Parsing log files

Exactly how you go about parsing a log file depends greatly on the specific format it takes. For log files with regular table format, tools like *cut*, *split*, *head*, and *tail* are likely to be useful. For all log files, *grep* is a great tool for finding and filtering contents of interest. For more complex processing tasks, you are likely to think of *sed*, *awk*, *perl*, or *python* as tools of choice.

For a good introduction to many of the text processing tools you are most likely to use in processing and analyzing log files, see David's IBM developerWorks tutorial on the GNU text processing utilities. A number of good higher-level tools also exist for working with log files, but these tools are usually distribution-specific and/or non-standard (but often Free Software) utilities.

Logging with syslogd and klogd

The daemon klogd intercepts and logs Linux kernel messages. As a rule, klogd will utilize the more general syslogd capabilities, but in special cases it may log messages directly to a file.

The general daemon syslogd provides logging for many programs. Every logged message contains at least a time and a hostname field and usually a program name field. The specific behavior of syslogd is controlled by the configuration file /etc/syslog.conf. Application (including kernel) messages may be logged to files, usually living under /var/log/ or remotely over a network socket.

Configuring /etc/syslog.conf

The file /etc/syslog.conf contains a set of rules, one per line. Empty lines and lines starting with a "#" are ignored. Each rule consists of two whitespace-separated fields, a selector field, and an action field. The selector, in turn, contains one of more dot-separated facility/priority pairs. A facility is a subsystem that wishes to log messages and can have the (case-insensitive) values: `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `mark`, `news`, `security`, `syslog`, `user`, `uucp`, and `local0` through `local7`.

Priorities have a specific order and matching a given priority means "this one or higher" unless an initial "=" (or "!=") is used. Priorities are, in ascending order: `debug`, `info`, `notice`, `warning` or `warn`, `err` or `error`, `crit`, `alert`, `emerg` or `panic` (several names have synonyms). `none` means that no priority is indicated.

Both facilities and priorities may accept the "*" wildcard. Multiple facilities may be comma-separated and multiple selectors may be semi-colon separated. For example:

```
# from /etc/syslog.conf
# all kernel mesages
kern.*                  -/var/log/kern.log
# `catch-all' logfile
*.=info;*.=notice;*.=warn;\
  auth,authpriv.none;\
  cron,daemon.none;\
  mail,news.none        -/var/log/messages
# Emergencies are sent to everybody logged in
*.emerg                 *
```

Configuring remote logging

To enable remote logging of syslogd messages (really application messages, but handled by syslogd), you must first enable the "syslog" service on both the listening and the sending machines. To do this, you need to add a line to each /etc/services configuration file containing something like:

```
syslog    514/UDP
```

To configure the local (sending) syslogd to send messages to a remote host, you specify a regular facility and priority but give an action beginning with an "@" symbol for the destination host. A host may be configured in the usual fashion, either /etc/hosts or via DNS (it need not be resolved already when syslogd first launches). For example:

```
# from /etc/syslog.conf
# log all critical messages to master.example.com
*.crit                @master.example.com
# log all mail messages except info level to mail.example.com
mail.*;mail.!=info    @mail.example.com
```

Rotating log files

Often you will not want to let particular log files grow unboundedly. The utility *logrotate* may be used to archive older logged information. Usually logrotate is run as a `cron` job, generally daily. logrotate allows automatic rotation, compression, removal, and mailing of log files. Each log file may be handled daily, weekly, monthly, or only when it grows too large.

The behavior of logrotate is controlled by the configuration file /etc/logrotate.conf (or some other file, if specified). The configuration file may contain both global options and file-specific options. Generally, archived logs are saved for a finite time period and are given sequential backup names. For example, one system of mine contains the following files due to its rotation schedule.

```
-rw-r-----  1 root adm 4135 2005-08-10 04:00 /var/log/syslog
-rw-r-----  1 root adm 6022 2005-08-09 07:36 /var/log/syslog.0
-rw-r-----  1 root adm  883 2005-08-08 07:35 /var/log/syslog.1.gz
-rw-r-----  1 root adm  931 2005-08-07 07:35 /var/log/syslog.2.gz
-rw-r-----  1 root adm  888 2005-08-06 07:35 /var/log/syslog.3.gz
-rw-r-----  1 root adm 9494 2005-08-05 07:35 /var/log/syslog.4.gz
-rw-r-----  1 root adm 8931 2005-08-04 07:35 /var/log/syslog.5.gz
```

Section 3. Packaging software

In the beginning was the tarball

For custom software distribution on Linux, there is actually much less needed than you might think. Linux has a fairly clean standard about where files of various types should reside and installing custom software, at its heart, need not involve much more than putting the right files in the right places.

The Linux tool *tar* (for "tape archive," though it need not, and usually does not, utilize tapes) is perfectly adequate to create an archive of files with specified filesystem

locations. For distribution, you generally want to compress a tar archive with gzip (or bzip2). See the final section of this tutorial on backup for more information on these utilities. A compressed tar archive is generally named with the extensions .tar.gz or .tgz (or .tar.bz2).

Early Linux distributions -- and some current ones like Slackware -- use simple tarballs as their distribution mechanism. For a custom distribution of in-house software to centrally maintained Linux systems, this continues to be the simplest approach.

Custom archive formats

Many programming languages and other tools come with custom distribution systems that are neutral between Linux distributions and usually between altogether different operating systems. Python has its *distutils* tools and archive format; Perl has CPAN archives; Java has .jar files; Ruby has gems. Many non-language applications have a standard system for distributing plug-ins or other enhancements to a base application as well.

While you can perfectly well use an package format like DEB or RPM to distribute a Python package for example, it often makes more sense to follow the packaging standard of the tool the package is created for. Of course, for system-level utilities and applications or for most compiled userland applications, that standard is the Linux distribution packaging standards. But for custom tools written in specific programming languages, something different might promote easier reuse of your tools across distributions and platforms (whether in-house or external users are the intended target).

The "big two" package formats

There are two main package formats used by Linux distributions: Redhat Package Manager (RPM) and Debian (DEB). Both are similar in purpose but different in details. In general, either one is a format for an "enhanced" archive of files. The enhancements provided by these package formats include annotations for version numbers, dependencies of one application upon other applications or libraries, human-readable descriptions of packaged tools, and a general mechanism for managing the installation, upgrade, and de-installation of packaged tools.

Under DEB files, the nested configuration file *control* contains most of the package metadata. For RPM files, the file *spec* plays this role. The full details of creating good packages in either format is beyond this tutorial, but we will outline the basics here.

What is in a .deb file?

A DEB package is created with the archive tool and cousin of tar, *ar* (or by some higher-level tool that utilizes ar). Therefore we can use ar to see just what is really

inside a .deb file. Normally we would use higher-level tools like *dpkg*, *dpkg-deb*, or *apt-get* to actually work with a DEB package. For example:

```
% ar tv unzip_5.51-2ubuntu1.1_i386.deb
rw-r--r-- 0/0      4 Aug  1 07:23 2005 debian-binary
rw-r--r-- 0/0   1007 Aug  1 07:23 2005 control.tar.gz
rw-r--r-- 0/0 133475 Aug  1 07:23 2005 data.tar.gz
```

The file debian-binary simply contains the DEB version (currently 2.0). The tarball data.tar.gz contains the actually application files -- executables, documentation, manual pages, configuration files, and so on.

The tarball control.tar.gz is the most interesting from a packaging perspective. Let us look at the example DEB package we chose:

```
% tar tvfz control.tar.gz
drwxr-xr-x root/root         0 2005-08-01 07:23:43 ./
-rw-r--r-- root/root       970 2005-08-01 07:23:43 ./md5sums
-rw-r--r-- root/root       593 2005-08-01 07:23:43 ./control
```

As you might expect, `md5sums` contains cryptographic hashes of all the distributed files for verification purposes. The file `control` is where the metadata lives. In some cases you might also find or wish to include scripts called *postinst* and *prerm* in control.tar.gz to take special steps after installation or before removal, respectively.

Creating a DEB control file

The installation scripts can do anything a shell script might. (Look at some examples in existing packages to get an idea.) But those scripts are optional and often not needed or included. Required for a .deb package is its control file. The format of this file contains various metadata fields and is best illustrated by showing an example:

```
% cat control
Package: unzip
Version: 5.51-2ubuntu1.1
Section: utils
Priority: optional
Architecture: i386
Depends: libc6 (>= 2.3.2.ds1-4)
Suggests: zip
Conflicts: unzip-crypt (<< 5.41)
Replaces: unzip-crypt (<< 5.41)
Installed-Size: 308
Maintainer: Santiago Vila <sanvila@debian.org>
Description: De-archiver for .zip files
 InfoZIP's unzip program. With the exception of multi-volume archives
 (ie, .ZIP files that are split across several disks using PKZIP's /& option),
 this can handle any file produced either by PKZIP, or the corresponding
 InfoZIP zip program.
 .
 This version supports encryption.
```

Basically, except the custom data values, you should make your control file look just like this one. For non-CPU specific packages -- either scripts, pure documentation,

or source code -- use `Architecture: all`.

Making a DEB package

Creating a DEB package is performed with the tool *dpkg-deb*. We cannot cover all the intricacies of good packaging, but the basic idea is to create a working directory, ./debian/, and put the necessary contents into it before running dpkg-deb. You will also want to set permissions on your files to match their intended state when installed. For example:

```
% mkdir -p ./debian/usr/bin/
% cp foo-util ./debian/usr/bin                # copy executable/script
% mkdir -p ./debian/usr/share/man/man1
% cp foo-util.1 ./debian/usr/share/man/man1   # copy the manpage
% gzip --best ./debian/usr/share/man/man1/foo-util.1
% find ./debian -type d | xarg chmod 755      # set dir permissions
% mkdir -p ./debian/DEBIAN
% cp control ./debian/DEBIAN   # first create a matching 'control'
% dpkg-deb --build debian      # create the archive
% mv debian.deb foo-util_1.3-1all.deb  # rename to final package name
```

More on DEB package creation

In the previous panel you can see that our local directory structure underneath ./debian/ is meant to match the intended installation structure. A few more points on creating a good package are worth observing.

- Generally you should create a file as part of your distribution called ./debian/usr/share/doc/foo-util/copyright (adjust for package name).
- It is also good practice to create the files ./debian/usr/share/doc/foo-util/changelog.gz and ./debian/usr/share/doc/foo-utils/changelog.Debian.gz.
- The tool *lintian* will check for questionable features in a DEB package. Not everything lintian complains about is strictly necessary to fix; but if you intend wider distribution, it is a good idea to fix all issues it raises.
- The tool fakeroot is helpful for packaging with the right owner. Usually a destination wants tools installed as root, not as the individual user who happened to generate the package (lintian will warn about this). You can accomplish this with:
  `% fakeroot dpkg-deb --build debian`

What is in an .rpm file?

RPM takes a slightly different strategy than DEB does in creating packages. Its configuration file is called *spec* rather than *control*, but the spec file also does more

work than a control file does. All the details of steps needed for pre-installation, post-installation, pre-removal, and installation itself, are contained as embedded script files in a spec configuration. In fact, the spec format even comes with macros for common actions.

Once you create an RPM package, you do so with the `rpm -b` utility. For example:

```
% rpm -ba foo-util-1.3.spec  # perform all build steps
```

This package build process does not rely on specific named directories as with DEB, but rather on directives in the more complex spec file.

Creating RPM metadata

The basic metadata in an RPM is much like that in a DEB. For example, foo-util-1.3.spec might contain something like:

```
# spec file for foo-util 1.3
Summary: A utility that fully foos
Name: foo-util
Version: 1.3
Release: 1
Copyright: GPL
Group: Application/Misc
Source: ftp://example.com/foo-util.tgz
URL: http://example.com/about-foo.html
Distribution: MyLinux
Vendor: Acme Systems
Packager: John Doe <jdoe@acme.example.com>

%description
The foo-util program is an advanced fooer that combines the
capabilities of OneTwo's foo-tool and those in the GPL bar-util.
```

Scripting in an RPM

Several sections of an RPM spec file may contain mini shell scripts. These include:

- `%prep`: Steps to perform to get the build ready such as clean out earlier (partial) builds. Often the following macro is helpful and sufficient:

```
%prep
%setup
```

- `%build`: Steps to actually build the tool. If you use the `make` facility, this might amount to:

```
%build
make
```

- `%install`: Steps to install the tool. Again, if you use `make` this might mean:

```
%install
make install
```

- `%files`: You *must* include a list of files that are part of the package. Even though your Makefile might use these files, the package manager program (rpm) will not know about them unless you include them here:

```
%files
%doc README
/usr/bin/foo-util
/usr/share/man/man1/foo-util.1
```

Section 4. Backup operations

About backup

The first rule about making backups is: Do it! It is all too easy in server administration -- or even just with Linux on the desktop -- to neglect backing up on a schedule suited to your requirements.

The easiest way to carry out backups in a systematic and schedules way is to perform them on a `cron` job. See the Topic 213 tutorial for a discussion of configuring *crontab*. In part, the choice of backup schedule depends on the backup tool and media you choose to use.

Backup to tape is a traditional technique and tape drives continue to offer the largest capacity of relatively inexpensive media. But recently, writeable or rewriteable CDs and DVD have become ubiquitous and will often make reasonable removable media for backups.

What to back up

A nice thing about Linux is that it uses a predicable and hierarchical arrangement of files. As a consequence, you rarely need to backup an entire filesystem hierarchy; most of a Linux filesystem hierarchy can be reinstalled from distribution media easily enough. In large environments, a master server image might be used to create a basic Linux system which can be customized by restoring a few selected files that were backed up elsewhere.

Basically, what you want backed up is /home/, /etc/, /usr/local/, and maybe /root/ and

/boot/. Often you will also want to backup some parts of /var/, such as /var/log/ and /var/mail/.

Backup with cp and scp

Perhaps the simplest way to perform a backup is with *cp* or *scp* and the `-r` (recurse) switch. The former copies to "local" media (but including NFS mounts), the latter can copy to remote servers in a securely encrypted fashion. For either, you need a mounted drive with sufficient space to accommodate the files you want to backup, of course. To gain any real hardware protection, you want the partition you copy to to be a different physical device than the drive(s) you are backing up from.

Copying with cp or scp can be part of an incremental backup schedule. The trick here is to use the utility *find* to figure out which files have been modified recently. Here is a simple example where we copy all the files in /home/ that have been modified in the lst day:

```
#!/bin/bash
# File: backup-daily.sh
# ++ Run this on a daily cron job ++
#-- first make sure the target directories exist
for d in `find /home -type d` ; do mkdir -p /mnt/backup$d ; done
#-- then copy all the recently modified files (one day)
for f in `find /home -mtime -1` ; do cp $f /mnt/backup$f  ; done
```

The `cp -u` flag is somewhat similar, but it depends on the continuity of the target filesystem between backup events. The find approach works fine if you change the mount point of /mnt/backup to a different NFS location. And the find system works equally well with scp once you specify the necessary login information to the remote site.

Backup with tar

Although cp and scp are workable for backup, generally tar sees wider use, being designed specifically for creating tape archives. Despite the origin of the name, tar is equally capable of creating a simple .tar file as raw data on a tape drive. For example, you might backup to a tape drive using a command like:

```
% tar -cvf /dev/rmt0 /home    # Archive /home to tape
```

To direct the output to a file is hardly any different:

```
% tar -cvf /mnt/backup/2005-08-12.tar /home
```

In fact, since gzip is streamable, you can easily compress an archive during creation:

```
% tar -cv /home | gzip - > /mnt/backup/2005-08-12.tgz
```

You can combine tar with find in the same ways shown for cp or scp. To list the files on a tape drive, you might use:

```
% tar -tvf /dev/rmt0
```

To retrieve a particular file:

```
% tar -xvf /dev/rmt0 file.name
```

Backup with cpio

The utility *cpio* is a superset of tar. It handles tar archives, but will also work with several other formats and has many more options built in. cpio comes with a huge wealth of switches to filter which files are backed up and even supports remote backup internally (rather than needing to pipe through scp or the like). The main advantage cpio has over tar is that you can both add files to archives and remove files back out.

Here are some quick examples of cpio:

- Create a file archive on a tape device: `% find /home -print |cpio -ocBv /dev/rmt0`.
- List the entries in a file archive on a tape device: `% cpio -itcvB < /dev/rmt0`.
- Retrieve a file from a tape drive: `% cpio -icvdBum file.name < /dev/rmt0`.

Backup with dump and restore

A set of tools named *dump* and *restore* (or with related names) are sometimes used to backup whole filesystems. Unfortunately, these tools are specific to filesystem types and are not uniformly available. For example, the original dump and restore are ext2/3-specific while the tools *xfsdump* and *xfsrestore* are used for XFS filesystems. Not every filesystem type has the equivalent tools and even if they do, switches are not necessarily uniform.

It is good to be aware of these utilities, but they are not very uniform across Linux systems. For some purposes -- like if you only use XFS partitions -- the performance of dump and restore can be a great boost over a simple tar or cpio.

Incremental backup with rsync

rsync is utility that provides fast incremental file transfer. For automated remote backups, rsync is often the best tool for the job. A nice feature of rsync over other tools is that it can optionally enforce two-way synchronization. That is, rather than simply backing up newer or changed files, it can also automatically remove locally deleted files from the remote backup.

To get a sense of the options, this moderately complex script (located at the rsync Web pages) is useful to look at:

```
#!/bin/sh
# This script does personal backups to a rsync backup server. You will
# end up with a 7 day rotating incremental backup. The incrementals will
# go into subdirs named after the day of the week, and the current
# full backup goes into a directory called "current"
# tridge@linuxcare.com
# directory to backup
BDIR=/home/$USER
# excludes file - this contains a wildcard pats of files to exclude
EXCLUDES=$HOME/cron/excludes
# the name of the backup machine
BSERVER=owl
# your password on the backup server
export RSYNC_PASSWORD=XXXXXX
BACKUPDIR=`date +%A`
OPTS="--force --ignore-errors --delete-excluded --exclude-from=$EXCLUDES
  --delete --backup --backup-dir=/$BACKUPDIR -a"
export PATH=$PATH:/bin:/usr/bin:/usr/local/bin
# the following line clears the last weeks incremental directory
[ -d $HOME/emptydir ] || mkdir $HOME/emptydir
rsync --delete -a $HOME/emptydir/ $BSERVER::$USER/$BACKUPDIR/
rmdir $HOME/emptydir
# now the actual transfer
rsync $OPTS $BDIR $BSERVER::$USER/current
```

Resources

Learn

- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- "Using the GNU text utilities" (developerWorks, March 2004) shows you how to use the GNU text utilities collection to process log files, documentation, structured text databases, and other textual sources of data or content.
- "Understanding Linux configuration files (developerWorks, December 2001)" explains configuration files on a Linux system that control user permissions, system applications, daemons, services, and other administrative tasks in a multi-user, multi-tasking environment.
- "Windows-to-Linux roadmap: Part 8. Backup and recovery" (developerWorks, November 2003) is a quick guide to Linux backup and recovery.
- *Installation Guide and Reference: Software Product Packaging Concepts* discusses the concepts of software packaging.
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Find sample Samba scripts at the rsync Web pages.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

David Mertz, Ph.D.
David Mertz is Turing complete, but probably would not pass the Turing Test. For more on his life, see his personal Web page. He's been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* .